A fix-and-optimize matheuristic for university timetabling

Lindahl, Michael; Sørensen, Matias; Stidsen, Thomas R.

Published in:
Journal of Heuristics

Link to article, DOI:
10.1007/s10732-018-9371-3

Publication date:
2018

Document Version
Peer reviewed version

Noname manuscript No.
(will be inserted by the editor)

A Fix-and-Optimize Matheuristic for University Timetabling

Michael Lindahl · Matias Sørensen · Thomas R. Stidsen

Received: date / Accepted: date

Abstract University course timetabling covers the task of assigning rooms and time periods to courses while ensuring a minimum violation of soft constraints that define the quality of the timetable. These soft constraints can have attributes that make it difficult for mixed-integer programming solvers to find good solutions fast enough to be used in a practical setting. Therefore, metaheuristics have dominated this area despite the fact that mixed-integer programming solvers have improved tremendously over the last decade. This paper presents a matheuristic where the MIP-solver is guided to find good feasible solutions faster. This makes the matheuristic applicable in practical settings, where mixed-integer programming solvers do not perform well. To the best of our knowledge this is the first matheuristic presented for the University Course Timetabling problem.

The matheuristic works as a large neighborhood search where the MIP solver is used to explore a part of the solution space in each iteration. The matheuristic uses problem specific knowledge to fix a number of variables and create smaller problems for the solver to work on, and thereby iteratively improves the solution. Thus we are able to solve very large instances and retrieve good solutions within reasonable time limits. The presented framework is easily extendable due to the flexibility of modeling with MIPs; new constraints and objectives can be added without the need to alter the algorithm itself. At the same time, the matheuristic will benefit from future improvements of MIP solvers.

The matheuristic is benchmarked on instances from the literature and the 2nd International Timetabling Competition (ITC2007). Our algorithm gives better solutions

M. Lindahl
Department of Management Engineering
Technical University of Denmark
E-mail: miclin@dtu.dk

M. Sørensen
MaCom A/S
E-mail: ms@macom.dk

T. Stidsen
Department of Management Engineering
Technical University of Denmark
E-mail: thst@dtu.dk

than running a state-of-the-art MIP solver directly on the model, especially on larger and more constrained instances. Compared to the winner of ITC2007, the matheuristic performs better. However, the most recent state-of-the-art metaheuristics outperform the matheuristic.

Keywords Matheuristics · Integer programming · University course timetabling

1 Introduction

University course timetabling is the problem of assigning courses to rooms and time periods. This should be done without violating a number of hard constraints that would make the timetable infeasible. For example, a teacher is only able to teach one class in a certain time period. Furthermore, a number of soft constraints should be obeyed as much as possible, because violating them would result in a timetable with undesired features. An undesired feature could e.g. be lectures of one course planned in different rooms during the week. The generating of high quality timetables automatically leads to better timetables in a production setting, and the automatization can help planners make timetables faster as they often have tight deadlines.

Timetabling differs between universities according to traditions and how their educations are structured. Many universities have curricula which consist of a number of courses taken by a group of students in a specific semester. This problem formulation is called the Curriculum-based Course Timetabling (CB-CTT). A formal description of this problem was made for the Second International Timetabling Competition in 2007 by Di Gaspero et al (2007) together with 21 benchmark instances from the University of Udine. This allowed researchers to compare their results on the same instances with roughly the same computational resources. The rules of the competition disallowed the use of external solvers which meant that all contestants used metaheuristics. A website has been made to keep track of the best known solutions and the best known bounds. The website is maintained by A. Bonutti, L. Gaspero and A. Schaerf and can be found at `http://tabu.diegm.uniud.it/ctt/`.

The purpose of this paper is to combine mixed-integer programming and heuristics to find good feasible solutions fast, taking advantage of the large improvements in mixed-integer programming solvers that have happened over the last decade as shown in Bixby (2012). This combination is commonly known as *matheuristics*. The matheuristic developed in this paper works as a large neighborhood search where the MIP solver is used to explore a part of the solution space in each iteration. The heuristic uses problem specific knowledge to fix a number of variables and create smaller problems for the solver to work on and thereby iteratively improves the solution. Thus, we are able to solve very large instances and retrieve good solutions within reasonable time limits. The presented framework is easily extendable due to the flexibility of modeling with MIPs; new constraints and objectives can be added without the need to alter the algorithm itself. At the same time, the matheuristic will benefit from future improvements in MIP solvers.

The paper is organized as follows: In section 2 we describe previous work. In section 3 the Curriculum-based Course Timetabling problem is defined together with the MIP formulation used. In section 4 the matheuristic is described. The computational results of the algorithm are shown in section 5. Finally, the conclusion and outlook of future directions are presented in section 6.

2 Previous work

The CB-CTT problem has received a lot of attention in the literature as the de-facto benchmarking problem within university timetabling. The winning heuristic of the original competition is described in Müller (2009). More recently, especially two heuristics have shown to perform well on the problem, see Abdullah and Turabieh (2012) and Kiefer et al (2014). As the algorithm presented in this paper is based on a MIP solver, we will focus on other MIP-based approaches in this section.

Integer programming has been applied to CB-CTT, but the models have proved difficult to solve due to the characteristics of the problem described by Burke et al (2008). In Hao and Benlic (2011) they successfully generate lower bounds by using MIP and a partition-based approach based on the "divide and conquer principle", and Cacchiani et al (2013) split the objective function into two parts formulated as MIPs that are solved separately by using a column generation procedure. In Lach and Lübbecke (2012) the problem is separated in two stages (two MIP models) that are exact with respect to the hard constraints when solved in sequence. They generate lower bounds and are also able to generate good feasible solutions. For a complete overview of the different methods applied to CB-CTT we recommend the overview by Bettinelli et al (2015). Hybridizing exact methods with heuristics is not a new idea. One of the earliest examples is the corridor method by Sniedovich and Voß (2006), where a corridor is made around the current solution to create a smaller search space. Within MIP Danna et al (2005) proposed a relaxation induced neighborhood search (RINS) by using the current incumbent together with the linear relaxation to search for improving solutions. New heuristics have been proposed, for example the local branching by Fischetti and Lodi (2003) who use the concept of hamming distance to create a branching based on the current solution. Within timetabling Avella et al (2007) applie a local search algorithm on a high-school timetabling problem and use a MIP solver to explore a very large neighborhood.

3 Curriculum-based Course Timetabling

In curriculum-based course timetabling (CB-CTT) the purpose is to assign a number of lectures to a time period and a room. To be able to compare results to the current state-of-the-art we use the formulation from the Second International Timetabling Competition ITC-2007 stated in Di Gaspero et al (2007).

A set of courses is given, and each course consists of a number of lectures that should be planned. A lecture should be assigned to a time period and a room without causing any conflicts. A set of time periods is given, and each one is associated with a day and a time, and two lectures of the same course can not take place at the same time or it will result in a conflict. A set of rooms is given, and only one lecture can take place in the same room in a time period, or it will result in a conflict. Each course also has a teacher assigned, and it will result in a conflict if a teacher has two lectures in the same time period. Furthermore, a set of curricula is also given. A curriculum consists of a set of courses, and courses from the same curriculum will conflict if they are planned in the same time period.

All lectures should be planned without conflicts, and the objective is to create a timetable that minimizes the violation of a number of soft constraints. The soft constraints define some attributes that we would like to avoid. Each one is associated

with a number of penalty points, and the goal is therefore to minimize the total sum of penalty points for the timetable. The following four soft constraints are defined:

RoomCapacity Each room has a capacity and each course has a number of students who attend the course. If a course is planned in a room with less capacity than the number of students attending the course, *one penalty point* is given for each student exceeding this number.

RoomStability A course consists of several lectures and it is desired that all of these are assigned to the same room. *one penalty point* is given for each additional room used.

MinimumWorkingDays To distribute the workload of a course throughout the week it is desired to spread the lectures out on several days. Each course has a number of minimum working days which should be respected. For each day below *five penalty points* are given.

CurriculumCompactness To avoid that students have idle time periods in the timetable it is desired that curricula are planned consecutively. If a course of a curriculum is planned in a time period where there is not another course from the same curriculum in the previous or in following time period, *two penalty points* are given.

3.1 Mixed Integer Programming Model

The proposed matheuristic works on top of a MIP model. Different models for the CB-CTT have been proposed in the literature and we refer to Bettinelli et al (2015) for an overview of these. We use the MIP model proposed in Lach and Lübbecke (2012). This model has shown good results on the ITC2007 benchmark instances, and we believe that this is currently the best model when the goal is to create feasible solutions. The approach consists of two stages solved sequentially. The first stage assigns time periods to the lectures, and the second stage then assigns the rooms. The decomposition is exact with respect to the hard constraints, meaning that no feasible solutions are lost.

The following sets are defined:

$\mathcal{C}$: Set of courses
$\mathcal{CU}$: Set of curricula
$\mathcal{P}$: Set of timeslots across the week
$\mathcal{D}$: Set of days of the week
$\mathcal{R}$: Set of rooms
$\mathcal{T}$: Set of teachers

The following parameters are defined:

$l(c)$: The number of lectures for course $c \in \mathcal{C}$
$mnd(c)$: The minimum working days for course $c \in \mathcal{C}$
$dem(c)$: The demand for course $c \in \mathcal{C}$ i.e. the number of students.
$cap(r)$: The capacity of room $r \in \mathcal{R}$
$\mathcal{C}(t)$ Set of courses where the teacher is $t \in \mathcal{T}$
$\mathcal{C}(cu)$ Set of courses included in curriculum $c \in \mathcal{CU}$

Furthermore, the following helper sets are used to formulate the model:

$\mathcal{C}_{\geq s} = \{c \in \mathcal{C} : dem(c) \geq s\}$, set of courses with a demand larger than or equal to $s \in \mathcal{S}$

$\mathcal{R}_{\geq s} = \{r \in \mathcal{R} : cap(r) \geq s\}$, set of rooms with capacity larger than or equal to $s \in \mathcal{S}$

3.1.1 Stage I

The first stage determines at what time periods each lecture should be taught. It does not assign any rooms, but keeps track of the available rooms to ensure that a feasible room assignment exists and calculates the violation of the *RoomCapacity* constraint. The Stage I therefore considers all soft constraints except for *RoomStability*. The decision variable is defined as,

$$x_{c,p} = \begin{cases} 1 & \text{if course } c \in \mathcal{C} \text{ is planned at period } p \in \mathcal{P} \\ 0 & \text{otherwise} \end{cases}$$

The following auxiliary variables are also needed: To keep track of the violation of *RoomCapacity*, the variable $y_{s,c,p}$ determines if course $c \in \mathcal{C}$ in period $p \in \mathcal{P}$ is planned in a room of size $s \in \mathcal{S}$ or smaller. The variable $z_{c,d}$ is equal to one if course $c \in \mathcal{C}$ is planned on day $d \in \mathcal{D}$. The variable w_c then counts the number of violations of the *MinimumWorkingDays* constraint for course $c \in \mathcal{C}$. The variable $r_{cu,p}$ is one if a course from curriculum $cu \in \mathcal{CU}$ is planned in period $p \in \mathcal{P}$. The violation of the *CurriculumCompactness* is then calculated by $v_{cu,p}$ that determines if there is an isolated lecture from curriculum $cu \in \mathcal{CU}$ in period $p \in \mathcal{P}$. The full model for stage I is shown in Model 1.

The objective (1a) calculates the violation of the three soft constraints. The constraints of the MIP are described in the following:

Constraints

- (1b) – A course $c \in \mathcal{C}$ should be assigned exactly L_c lectures.
- (1c) – At given period $p \in \mathcal{P}$ it is not possible to assign more courses than available rooms.
- (1d) – If a room size is assigned to a course, the course should be assigned to that time period.
- (1e) – Ensures that we do not assign more courses to a certain room size than rooms available of that size.
- (1f) – Calculates if a course $c \in \mathcal{C}$ is planned on day $d \in \mathcal{D}$
- (1g) – Calculates the violation of the *MinimumWorkingDay* constraint.
- (1h) – Calculates if a curriculum is planned in a time period and ensures that only one course from the same curriculum is planned.
- (1i) – Calculates the *CurriculumCompactness* violation.
- (1j) – Ensures that only one course with the same teacher is planned at the same period.

3.1.2 Stage II

The Stage II model assigns a room to each lecture based on the solution of Stage I while minimizing *RoomStability*. The decision variable is the following:

$$u_{c,p,r} = \begin{cases} 1 & \text{if course } c \in \mathcal{C} \text{ is planned in room } r \in \mathcal{R} \text{ at period } p \in \mathcal{P} \\ 0 & \text{otherwise} \end{cases}$$

$$\min \quad \sum_{p\in\mathcal{P},s\in\mathcal{S},c\in\mathcal{C}_{\geq s}} obj_{s,c,p}\cdot y_{s,c,p} + \sum_{c\in\mathcal{C}} 5\cdot w_c + \sum_{cu\in\mathcal{CU},p\in\mathcal{P}} 2\cdot v_{cu,p} \tag{1a}$$

$$\text{s.t.} \quad \sum_{p\in\mathcal{P}} x_{c,p} = L(c) \qquad \forall c\in\mathcal{C} \tag{1b}$$

$$\sum_{c\in\mathcal{C}} x_{c,p} \leq |\mathcal{R}| \qquad \forall p\in\mathcal{P} \tag{1c}$$

$$x_{c,p} - y_{s,c,p} \geq 0 \qquad \forall c\in\mathcal{C}, p\in\mathcal{P}, s\in\mathcal{S} \tag{1d}$$

$$\sum_{c\in\mathcal{C}_{\geq s}} (x_{c,p} - y_{s,c,p}) \leq |\mathcal{R}_{\geq s}| \qquad \forall s\in\mathcal{S}, p\in\mathcal{P} \tag{1e}$$

$$\sum_{p\in\mathcal{P}} x_{c,p} - z_{c,d} \geq 0 \qquad \forall c\in\mathcal{C}, d\in\mathcal{D} \tag{1f}$$

$$\sum_{d\in\mathcal{D}} z_{c,d} + w_c \geq mnd(c) \qquad \forall c\in\mathcal{C} \tag{1g}$$

$$\sum_{c\in\mathcal{C}(cu)} x_{c,p} - r_{cu,p} = 0 \qquad \forall cu\in\mathcal{CU}, p\in\mathcal{P} \tag{1h}$$

$$-r_{cu,p-1} + r_{cu,p} - r_{cu,p+1} - v_{cu,p} \leq 0 \qquad \forall cu\in\mathcal{CU}, p\in\mathcal{P} \tag{1i}$$

$$\sum_{c\in\mathcal{C}(t)} x_{c,p} \leq 1 \qquad \forall t\in\mathcal{T}, p\in\mathcal{P} \tag{1j}$$

$$x_{c,p}\in\mathbb{B} \qquad \forall c\in\mathcal{C}, p\in\mathcal{P} \tag{1k}$$

$$y_{s,c,p}\in\mathbb{B} \qquad \forall s\in\mathcal{S}, c\in C_{\geq s}, p\in\mathcal{P} \tag{1l}$$

$$w_c\in\mathbb{Z}_+ \qquad \forall c\in\mathcal{C} \tag{1m}$$

$$z_{c,d}\in\mathbb{B} \qquad \forall c\in\mathcal{C}, d\in\mathcal{D} \tag{1n}$$

$$v_{cu,p}\in\mathbb{B} \qquad \forall cu\in\mathcal{CU}, p\in\mathcal{P} \tag{1o}$$

$$r_{cu,p}\in\mathbb{B} \qquad \forall cu\in\mathcal{CU}, p\in\mathcal{P} \tag{1p}$$

Model 1 The MIP model for the first stage.

The assignments of times is given from the previous stage as $x^*_{c,p}$. If a course is not assigned to a time period, no room should be assigned. The violation of room capacity is given from the variables $y^*_{s,c,p}$. This means that if a lecture has been assigned to a smaller room in Stage I this should also be the case in the second stage.

Given a course $c\in\mathcal{C}$, a time period $p\in\mathcal{P}$ and a room $r\in\mathcal{R}$ then the assignment is invalid, i.e. $u_{c,p,r}=0$, if one of these three conditions is satisfied:

- If a course c is not assigned to a time period p then no room should be assigned.
  - $x^*_{c,p}=0$
- If there is no violation of the capacity in Stage I then there should not be a violation in Stage II.
  - $y^*_{s,c,p}=0$ and $dem(c) > cap(r)$
- If the capacity is exceeded in Stage I, then the same violation should occur in Stage II.
  - $y^*_{s,c,p}=1$
  - $dem(c)\leq cap(r)$
  - $cap(r) = \max_{\hat{r}\in\mathcal{R}}\{cap(\hat{r}) : cap(\hat{r}) < dem(c))\}$

Furthermore, the variable $y_{c,r}$ states whether course $c \in \mathcal{C}$ takes place in room $r \in \mathcal{R}$ at least once. This is used to calculate the violation of the *RoomStability* constraint. The full model of Stage II is shown in Model 2.

$$\min \quad \sum_{c\in\mathcal{C}, r\in\mathcal{R}} y_{c,r} \tag{2a}$$

$$\text{s.t.} \quad \sum_{p\in\mathcal{P}} u_{c,p,r} - |\mathcal{P}| \cdot y_{c,r} \leq 0 \quad \forall c \in \mathcal{C}, r \in \mathcal{R} \tag{2b}$$

$$\sum_{r\in\mathcal{R}} u_{c,p,r} = 1 \quad \forall c \in \mathcal{C}, p \in \mathcal{P} : x^*_{c,p} = 1 \tag{2c}$$

$$\sum_{c\in\mathcal{C}, p\in\mathcal{P}} u_{c,p,r} \leq 1 \quad \forall r \in \mathcal{R} \tag{2d}$$

$$y_{c,r} \in \mathbb{B} \quad \forall c \in \mathcal{C}, r \in \mathcal{R} \tag{2e}$$

$$u_{c,p,r} \in \mathbb{B} \quad \forall c \in \mathcal{C}, p \in \mathcal{P}, r \in \mathcal{R} \tag{2f}$$

Model 2 The MIP model for the second stage.

The objective (2a) is the violation of the *RoomStability* objective. The constraints are the following:

Constraints

(2b) – Calculates the violation of the *RoomCapacity* constraint.
(2c) – Ensures that all lectures get assigned to a room.
(2d) – Ensures that no more than one lecture is assigned to the same room.

4 Fix-and-Optimize Matheuristic

A natural way of finding good solutions to a problem is to iteratively improve a bad solution. This is done by creating a neighborhood that can be explored around the solution, like proposed in the Large Neighborhood Search (LNS) algorithm (Shaw (1998)). An example of a matheuristic is the corridor method proposed by Sniedovich and Voß (2006). In the corridor method an exact method that can solve small instances of a problem, but which is not applicable to large instances, is used. A corridor around the current solution is created, and this results in a smaller neighborhood which can then be explored by using the exact method. Our matheuristic build on a similar approach, as we have a mixed-integer model that can solve small instances to optimality, but struggles with large instances. In each iteration the MIP solver is used to explore a large neighborhood defined by fixing a subset of the variables. This results in a new MIP that we call the *subproblem*, and the solver is then used as a black-box to search for improving solutions within this. This is the *fix-and-optimize* aspect of the matheuristic. This also implies that if the original model is changed, e.g. a constraint is added or removed, the neighborhoods are not affected as the same model is used. This can sometimes be a problem in move-based heuristics where a new constraint can make certain moves obsolete as they depend too much on special characteristics of the solution.

An important aspect of implementing this algorithm is that the model is only built from scratch once by the MIP-solver. As the same model instance is used throughout the algorithm, and due to the way the fixing of the variables is defined, the solution found in the previous iteration of the algorithm is always feasible with regard to the fixing of the variables in the next iteration. For most MIP solvers, this means that the solve operation in each iteration is automatically warm-started from the previously found solution, which is a great benefit. For the MIP solver used in these experiments (Gurobi) this is certainly the case.

In Caserta and Voss (2010) they put metaheuristics in two classes. The first is the *model based heuristics* where a new solution is found by using a model. Our algorithm falls in the second category called *method based heuristic* where the underlying model is still the same, but the neighborhood is defined by how the MIP solver explores the neighborhood.

The proposed method is applied to Stage I of the problem as experiments have shown that the majority of the solution time is spent at this stage. The resulting solution is then handed to the Stage II model to find a solution for the entire problem.

The algorithm is described in Algorithm 4.1. A key part of the algorithm is to determine which variables to fix to create subproblems that are easier to solve than the full problem, while still leading to improving solutions. This part will be explained in Section 4.2.

Algorithm 4.1 Fix-and-Optimize Matheuristic

input:
    problem instance
    Set of neighborhoods N and initial sizes S
output:
    Solution
$x^*_{c,p} \leftarrow$ Create Initial solution
Fix all decision variables
while stopping criteria not met **do**
    Pick neighborhood $n \in N$
    $X(R)$ = NeighborhoodCreator(n) ▷ Find connected decision variables
    Unfix variables $X(R)$
    $x^*_{c,p} \leftarrow$ Optimize subproblem
    Update S_n ▷ Use feedback from solver to update parameters
end while
Solve StageII($x^*_{c,p}$)

4.1 Initial solution

Before decisions variables can be fixed, a feasible initial solution is needed. As shown in Lach and Lübbecke (2012) the problem can quickly be solved when the soft constraint *CurriculumCompactness* is removed. Moreover, the objective *MinimumWorkingDays* also introduces new auxiliary variables, and removing this makes the problem easier. Creating a start solution is therefore done by removing those two objectives: *CurriculumCompactness* and *MinimumWorkingDays*. To avoid spending too much time in this step the solver is set to return the first found solution.

4.2 Neighborhoods

A neighborhood defines which part of the solution space should be explored in an iteration. As mentioned, this is done by choosing a subset of variables that should be unfixed thus allowing the values to be changed.

The neighborhood should choose variables where it is likely that a change will result in an improved solution. If the resulting subproblem is too constrained by the fixed variables, then no new solutions will be found. At the same time it should also be possible for the solver to find an improving solution which depends on how difficult the subproblem is to solve. If the neighborhood is too difficult to solve within the time limit of each iteration, no improvement will be made.

There is no exact way to predict how difficult a MIP model is to solve, but as mentioned by Vielma (2015) the size of the model has a high impact. A simple measure which also is easy to calculate, is the number of decision variables. We therefore define the *size* of a neighborhood as the number of unfixed decision variables. In each iteration we have a neighborhood size, $S \in \mathbb{Z}_+$, which will determine how many variables to pick.

It is also important to choose decision variables that are connected in the sense that if one of them changes value, the others are likely to change value as well. An example is that if an assignment of a course in a curriculum is moved to a new time period, it is likely to affect other courses in that same curriculum, as they may need to be assigned to a new time period to avoid a conflict.

We define a resource r as an entity associated to a set of decision variables. We define the variables associated with resource r as $X(r)$. A resource could for example be a course where the associated decision variables are the ones that affect that course. i.e. to a course $c' \in \mathbb{C}$ the corresponding decision variables are

$$X(c') = X_{c,p} : c = c', p \in \mathbb{P}$$

To pick resources that are connected we create a *score* function to measure how well a specific resource is connected to a set of other resources. A high number means that if a decision variable from the set of resources is changed, it will likely result in a change in the given resource.
The score of resource r related to the list of resources R is defined the following way:

$$Score(R, r) \in \mathbb{R}_0^+$$

The algorithm uses a greedy heuristic to find connected resources. It starts with a list of resources of the same type, for example all courses. It then picks the first at random and then iteratively adds more depending on which one has the largest score. This is repeated until the desired size is reached. The algorithm is shown in Algorithm 4.2.

Three different neighborhoods are defined. The purpose of this is to make sure that it is possible to reach different parts of the solution space. The neighborhoods are put into two categories, static and dynamic. The static neighborhoods look at the problem formulation to pick variables where the dynamic looks at the current solution too. The three neighborhoods are: *Curricula*, *Courses* and *Assignments*.

Algorithm 4.2 NeighborhoodCreator

input: List of resources R, Neighborhood Size S
output: List of decision variables $\tilde{X}$
$\bar{R} := \{Random_{r \in R}\}$ ▷ Pick a random resource
while $|X(\bar{R})| < S$ **do** ▷ Add more until desired size is reached
  $r = \arg\max_{r \in R} Score(\{\bar{R}, r\})$ ▷ Find resource with highest score
  $\bar{R} := \bar{R} \cup r$ ▷ Add resource
end while
return $X(\bar{R})$ ▷ Return corresponding decision variables

Curricula The curricula neighborhood is static and chooses the decision variables of courses that belong to a set of curricula. Looking at optimal solutions, the objective *CurriculumCompactness* is usually contributing to a large part of the penalty. The purpose of this neighborhood is to make sure that the parts of the solution space that can lead to lowering this are explored. The score is calculated by choosing curricula with overlapping courses as moving one lecture will influence both curricula. The resources are therefore curricula $\mathcal{CU}$ and the score function is the number of courses in common:

$$Score(cu', \mathcal{CU}') = \left|c \in cu'\right| : c \in \mathcal{CU}'$$

Courses This neighborhood finds courses that have similar numbers of students attending and is therefore static. This is to be able to swap courses that use similar rooms. The resource is courses $\mathcal{C}$ with the following score function that measures the difference from the average number of students:

$$Score(c', \mathcal{C}') = -\left|dem(c') - \frac{1}{|\mathcal{C}'|}\sum_{c \in \mathcal{C}'} dem(c)\right|$$

Assignments This dynamic neighborhood looks at the assignments in the current best solution and looks at how much the assignment violates the soft constraints . The purpose of this is to fix undesired assignments that result in lowering the objective. Each assignment is therefore associated with the amount of violation in which it results.

$$Score(y'_{s',c',p'}, Y) = obj_{s',c',p'} + 5 \cdot w'_c + \sum_{p,cu \in \mathcal{CU}: c \in \mathcal{CU}} 2 \cdot r_{cu,p}$$

4.3 Choosing neighborhood

In each iteration it needs to be decided which neighborhood to use. This is done in a random manner by selecting what neighborhood to use from a uniform distribution.

4.4 Adaptive Neighborhood Size

Finding the right size of the neighborhoods is important. We therefore continually adjust the neighborhood size S to ensure that we are solving neither too easy or too difficult subproblems.

A way to measure the hardness of a subproblem is to look how far from optimum we are when the time limit of an iteration is reached. This is done by looking at the relative gap between the lower bound and the incumbent for each subproblem. We define two parameters *MinGap* and *MaxGap* and aim at creating subproblems that are solved within this gap. If the gap is less than *MinGap* the neighborhood size of that neighborhood is increased, and in the same way decreases it if it is larger than *MaxGap*.

The increase or decrease of the neighborhood size is defined in terms of the parameter *Decay*. Because each neighborhood is different, we have a size S for each of the used neighborhoods. The adaptive algorithm for updating the sizes is shown in Algorithm 4.3.

Algorithm 4.3 UpdateNeighborhoodsize

```
1: if MipGap > MaxGap then                    ▷ Subproblem is too difficult
2:     Size *= 1 - Decay                      ▷ Decrease size
3: else if MipGap < MinGap then               ▷ Subproblem is too easy
4:     Size *= 1 + Decay                      ▷ Increase size
5: end if
```

Smoothing The amount of decision variables is not enough to predict how difficult a subproblem is. Therefore, a lot of fluctuation in the MIP Gap happens between iterations for the same number of decision variables. To handle this we smooth the MIP Gap by using *exponential smoothing*, which makes it less fluctuated by averaging the MIP Gap with its previous value. This is done in the following way: Let $MipGap_i$ be the gap in iteration i, the smoothed gap, denoted $\widetilde{MipGap}_i$, is then calculated in the following way, where α is the *smoothing factor*.

$$\widetilde{MipGap}_1 = MipGap_1$$
$$\widetilde{MipGap}_i = \alpha \cdot MipGap_i + (1-\alpha) \cdot \widetilde{MipGap}_{i-1}, \quad i > 1$$

An example of how the size of the neighborhood adapts when using the MipGap is seen in Figure 1. It is seen that there is a lot of fluctuation in the MIPGap, but it is smoothed by the *exponential smoothing*.

5 Computational results

To evaluate the algorithm we use both the instances from the ITC2007 competition and the much larger instances from University of Erlangen. All data sets are available from `http://tabu.diegm.uniud.it/ctt/`. We will compare the algorithm to running the MIP solver directly on the CB-CTT model. As the goal is to provide a tool for practical

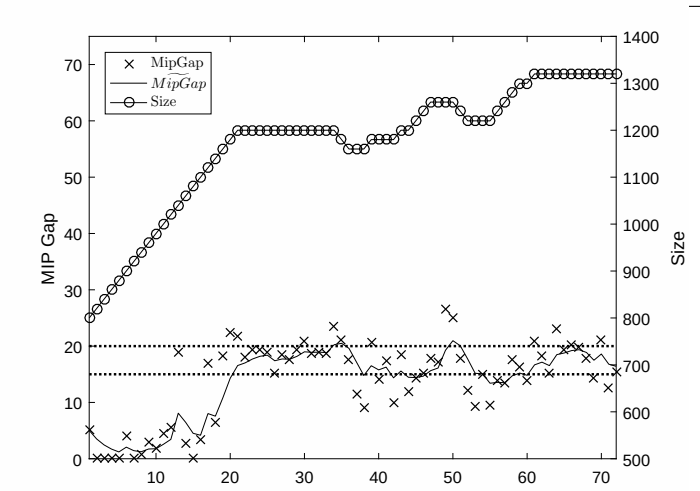

Fig. 1 This figure illustrates the size of the neighborhood and the MipGap in each iteration. It can be seen that the size of the neighborhood is gradually increased until it is stabilized and then increased in the end again.

Neighborhood	Curricula	Courses	Assignments
Size	1000	2500	60%

Table 1 The initial sizes of the neighborhoods. Two of them are absolute number of decision variables and the other are a percentage of the total amount.

timetabling, we will compare to the state-of-the-art in metaheuristic, both the winner from the original ITC2007 competition as well as newer algorithms proposed in the literature.

All experiments were run on a 64 bit Windows machine with a 4.00GHz CPU and 32GB of memory. To solve the integer programs we use Gurobi 6.5.0 running with standard parameters except for `MIPFocus = 1` to tell the solver to focus on finding feasible solutions instead of proving optimality. The solver is only allowed to use one thread. The allowed time limit is determined by using the benchmark tool provided for the ITC2007 competition. This means that *one CPU time unit* corresponds to 260 seconds. We use the same fraction on time for Stage I and II as proposed by Lach and Lübbecke (2012), where approximately 75% of the time is used in Stage I. This means that Stage I is given 210 seconds and Stage II 50 seconds.

The time limit for each iteration in the matheuristic is set to *2 seconds*. For the adaptive part the following parameters are used: `GapMin` $= 15\%$, `GapMax` $= 20\%$, $\alpha = 0.3$ and `Decay` $= .02$. The initial sizes of the neighborhoods are shown in Table 1. These are either an absolute number of decision variables or as an percentage of the total amount in the given instance.

All results are shown as the average over 10 runs with different random seeds to the matheuristic and the MIP solver.

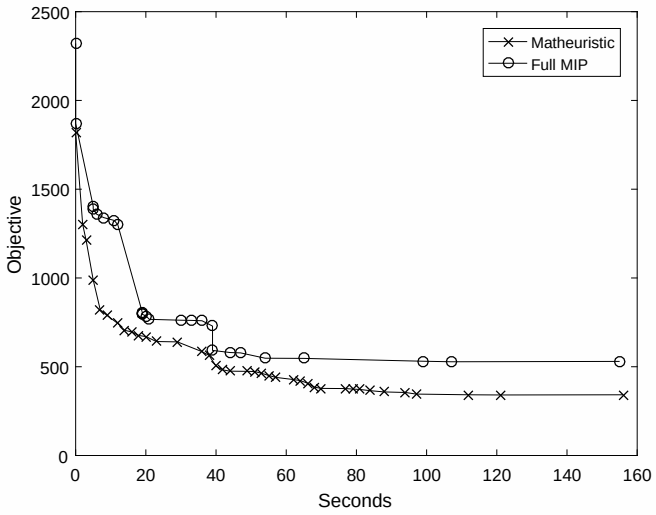

Fig. 2 Solving Comp12. It can be seen that the matheuristic makes many small improvements compared to the full MIP that hits some large improvement less frequently.

5.1 Comparison of the neighborhoods and the full MIP

To see the effects of the three neighborhoods, a comparison is made between the individual performance of each neighborhood, the full matheuristic and solving the full model directly by using a state-of-the-art MIP solver. In Table 2 we show the results on solving Stage I of the model. It is seen that each neighborhood by itself does not perform well, especially the assignment neighborhood performs very badly. The *courses* neighborhood works best by itself. It is seen that the performance is improved when two neighborhoods work together, and the algorithm works best when all three neighborhoods are used.

Looking at how the solution improves over time, illustrated in Figure 2, it is seen that the profiles differ. The matheuristic makes many small improvements, and the full MIP hits some larger improvements, but these occur less frequently.

5.2 Adaptive Neighborhood Sizes

To see the impact of the adaptive neighborhood sizes described in Section 4.4, tests are run on the Stage I model with and without the adaptive part. The results are shown in Table 3. It is seen that the adaptive neighborhood sizes have a large impact on the results, finding the best solution for 19 of 21 instances. How the adaptive part affects the neighborhood sizes can be seen in Figure 3. The two plots show neighborhood sizes and the smoothed MIP gap respectively for the three neighborhoods. It is seen that with the fixed neighborhood sizes, one of the neighborhoods always has a 100% MIP gap, and one always have 0%. With the adaptive part these neighborhood sizes are adjusted so the MIP gap falls between the min. and max. gap of 15% and 20%.

Instance	Full MIP	Cou	Cur	Assi	Cou,Cur	Cou,Assi	Cur,Assi	All
comp01	**4.0**	**4.0**	**4.0**	27.3	**4.0**	**4.0**	**4.0**	**4.0**
comp02	50.7	55.4	57.6	159.2	**39.2**	58.9	49.0	45.8
comp03	87.2	98.4	77.8	131.7	**72.1**	92.1	76.9	73.4
comp04	**35.0**	35.3	**35.0**	90.3	**35.0**	**35.0**	**35.0**	**35.0**
comp05	392.2	567.1	**369.6**	687.8	378.1	517.1	379.6	**369.6**
comp06	47.2	57.6	66.9	71.6	43.5	45.6	46.6	**42.4**
comp07	**6.2**	15.1	67.2	23.6	8.0	9.7	10.4	7.8
comp08	**37.0**	37.6	37.0	81.8	**37.0**	38.2	**37.0**	**37.0**
comp09	100.1	100.6	106.9	138.2	99.7	99.8	99.6	**99.1**
comp10	**4.4**	22.1	22.8	59.0	11.4	17.1	11.4	10.2
comp11	**0.0**	**0.0**	**0.0**	**14.2**	**0.0**	**0.0**	**0.0**	**0.0**
comp12	462.5	542.0	366.8	531.5	364.0	418.3	360.4	**359.6**
comp13	**61.4**	68.7	65.5	141.7	61.7	65.2	62.0	61.7
comp14	**51.6**	66.1	61.6	84.8	57.7	59.3	56.5	55.1
comp15	87.2	97.1	76.5	127.6	78.5	92.0	80.3	**73.7**
comp16	**22.1**	36.5	59.9	56.4	29.6	32.8	25.3	26.9
comp17	76.8	79.0	83.5	100.5	**70.7**	75.0	73.4	71.5
comp18	78.9	89.7	76.4	106.7	75.4	76.9	80.7	**71.9**
comp19	**58.1**	70.8	59.1	179.7	59.4	69.0	59.2	59.8
comp20	18.4	41.9	55.9	48.5	16.3	24.1	20.4	**17.7**
comp21	107.2	118.1	107.1	162.1	97.7	105.5	99.5	**95.3**
Avg. Rank	3.0	5.4	4.4	6.7	2.1	4.1	2.9	**1.7**

Table 2 Comparison of all combinations of the three neighborhoods Courses, Curriculum, Assignments and the full model on stage I. The best value for each instance is marked with bold. It can be seen that overall the heuristic helps the solver to find better solutions.

Instance	No Adaptive	Adaptive
comp01	**4.0**	**4.0**
comp02	51.4	**45.8**
comp03	81.9	**73.4**
comp04	**35.0**	**35.0**
comp05	405.6	**369.6**
comp06	43.8	**42.4**
comp07	8.2	**7.8**
comp08	**37.0**	**37.0**
comp09	101.6	**99.1**
comp10	14.5	**10.2**
comp11	**0.0**	**0.0**
comp12	384.9	**359.6**
comp13	**61.2**	61.7
comp14	56.6	**55.1**
comp15	82.6	**73.7**
comp16	28.1	**26.9**
comp17	73.5	**71.5**
comp18	80.5	**71.9**
comp19	**59.3**	59.8
comp20	20.6	**17.7**
comp21	96.5	**95.3**
Avg. Rank	1.7	1.1

Table 3 The matheuristic with and without the adaptive part on stage I. It is seen that the adaptive part greatly improves the algorithm and finds the best solution on 19 out of 21 instances.

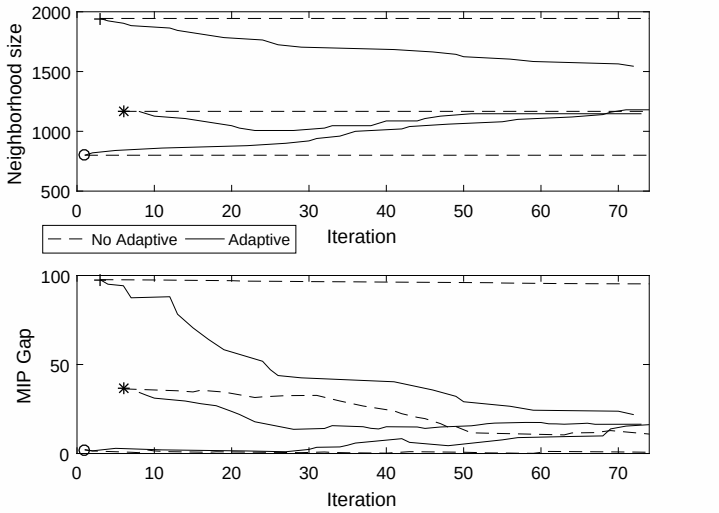

Fig. 3 The change in neighborhood size and MIP gap for the three neighborhoods with and without the adaptive part. Without the adaptive part two of the MIP Gap's stay at 0% and 100% respectively. The adaptive part changes the size of the neighboorhods so the MIP gaps stay within 15% to 20%.

5.3 Comparison with State of the art metaheuristics

To compare with the current state of the art of finding high quality solutions to CB-CTT we compare with Müller (2009), the winner of the ITC2007 contest. We also compare with two more recent algorithms from the literature, which have shown good results, that is a Tabu-based memetic from Abdullah and Turabieh (2012) and a LNS from Kiefer et al (2014). The results are shown in Table 4.

The matheuristic performs better than the original winner from ITC2007, Müller, but worse than the LNS and the Tabu-based memetic. On one instance the matheuristic finds the best solution.

5.3.1 Very large instances

To see how the algorithm performs on very large instances we use the ones from University of Erlangen. These instances have around six times as many decision variables at the first stage and are far more constrained. Because these instances are much bigger, the timelimit is set to 10 CPU time units. The results can be seen in Table 5. For two of the instances the solver applied to the full MIP is not able to find a feasible solution, and in all cases the matheuristic is able to find better solutions. Wbest show the currently best known solution. An example of a run is seen in Figure 4 which shows that the solver has difficulties in improving the solution for the full MIP. This shows that the matheuristic is more stable in finding feasible solutions and improving them afterwards.

Instance	Kiefer et al (2014)	Abdullah and Turabieh (2012)	Müller (2009)	MH
comp01	**5.0**	**5.0**	**5.0**	12.0
comp02	41.9	**36.4**	61.3	49.5
comp03	**72.8**	74.4	94.8	74.5
comp04	**35.2**	38.5	42.8	38.5
comp05	**306.3**	314.5	343.5	373.5
comp06	**48.1**	45.3	56.8	58.3
comp07	15.3	**12.0**	33.9	35.0
comp08	**40.6**	40.8	46.5	49.7
comp09	102.4	108.4	113.1	**100.5**
comp10	13.3	**8.4**	21.3	25.7
comp11	**0.0**	**0.0**	**0.0**	6.5
comp12	323.9	**320.3**	351.6	360.7
comp13	**63.8**	64.3	73.9	69.0
comp14	**56.1**	64.4	61.8	56.9
comp15	73.8	**72.7**	94.8	74.5
comp16	34.8	**23.7**	41.2	37.1
comp17	**73.0**	76.4	86.6	86.1
comp18	**66.5**	75.6	91.7	72.9
comp19	**64.6**	66.8	68.8	64.8
comp20	24.0	**13.5**	34.3	34.3
comp21	**95.3**	100.7	108.0	103.8
Avg. rank	1.4	1.8	3.3	3.1

Table 4 Comparison with the matheuristic including stage II with state of the art metaheuristics. The running time is 1 CPU Time unit. The bold results are the best result for that instance. The matheuristic (MH) performs better than the initial winner of the competition but not better than more recent versions.

Instance	Wbest	Full MIP	Matheuristic
erlangen2011_2	4670	-	**5,956.0**
erlangen2012_1	5716	19,067.0	**9,648.8**
erlangen2012_2	8813	-	**15,059.8**
erlangen2013_1	5476	20,467.0	**10,052.0**
erlangen2013_2	8150	16,308.0	**11,120.4**
erlangen2014_1	5981	15,765.0	**8,372.0**
Avg. Rank		2	1

Table 5 Comparison of the matheuristic on 6 very large instances running with 10 CPU time units. The results are for the full model (StageI+II) and Wbest is the currently best known solution. The matheuristic outperforms the full mip consistently and is able to find solutions where the solver can not on two of the instances.

6 Conclusion

We have proposed a matheuristic that works as a framework on top of a MIP model. The approach makes it possible to find good solutions even on larger and more constrained instances.

The algorithm has been applied to the standard benchmark datasets for Curriculum-based University Course Timetabling. To the best of our knowledge, this is the first paper which describes a matheuristic for this problem.

Computational results have shown that the matheuristic is better at finding feasible solutions than solving the original MIP with a MIP solver.

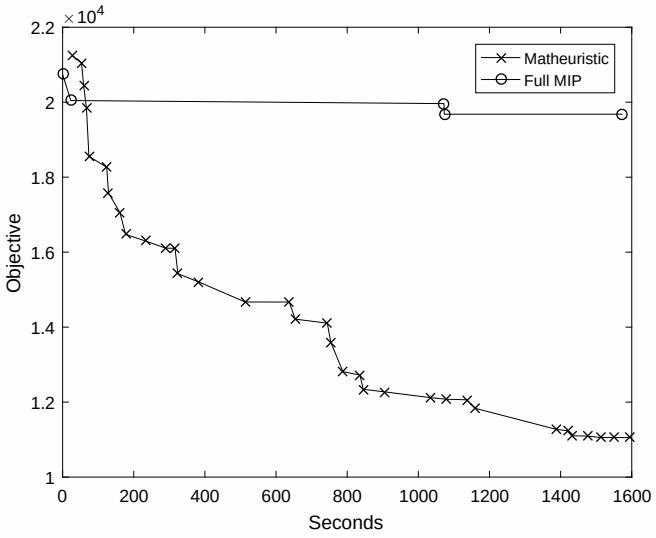

Fig. 4 Solving the very large instance Erlangen2012_1. The matheuristic finds a worse start solution but is able to improve it compared to the solver on the full model that can not find any improving solution.

Compared to metaheuristics, which is the most used method for practical timetabling, the algorithm performs well. On average, it performs better than the original winner from ITC2007, but not better than two more recently proposed heuristics. Research in heuristics for timetabling has a long history, and more research is needed before matheuristics can catch up. The proposed method can, however, utilize future improvements of both solvers and models, and especially the solvers have improved tremendously over the last decade Bixby (2012). Furthermore, the matheuristic algorithm has the advantage of being flexible when it comes to adding new constraints and objectives. The authors therefore believe that matheuristics and mathematical programming will become a major approach for practical timetabling.

6.1 Outlook

There are a lot of opportunities for future research in the area of matheuristics, aso applied to CB-CTT. An important aspect is to make sure that the computational resources are used to solve the right subproblems. In the following we propose three areas as subjects for future research.

Neighbourhoods It would be interesting to look more into what the characteristics of a good neighborhood are. This would also make it easier to adapt the algorithm to new problems. Another interesting thing would be to generalize it to generic MIPs. This would require neighborhoods with more generic definitions, as opposed to using problem-specific knowledge.

More Adaptive parameters The time limit in each iteration and the min. and max. MIP gap is manually set. There might be a potential in making them adaptive. So

that for example in the beginning there is a short time limit and high MIP gap, and in the later iterations, where the solution is closer to optimality, the MIP gap in each iteration is smaller, and the time limit is longer.

Parallelization Because of the fixed variables many of the neighborhoods are not connected and can therefore be solved in parallel. This can lead to further speed up by exploiting multiple processors. This would, however, require the processors to somehow share an instance of the MIP model, a problem to which there is no obvious solution in our opinion.

Acknowledgements The authors would like to thank the organizers of ITC-2007 for providing a formal problem description of CB-CTT as well as benchmark instances. The authors would also like to thank Alex Bonutti, Luca Di Gaspero and Andrea Schaerf for creating and maintaining the website for instances and solutions to CB-CTT.

References

Abdullah S, Turabieh H (2012) On the use of multi neighbourhood structures within a tabu-based memetic approach to university timetabling problems. information sciences 191:146–168

Avella P, D'Auria B, Salerno S, Vasilev I (2007) A computational study of local search algorithms for italian high-school timetabling. Journal of Heuristics 13:543–556

Bettinelli A, Cacchiani V, Roberti R, Toth P (2015) An overview of curriculum-based course timetabling. TOP pp 1–37

Bixby RE (2012) Optimization Stories, 21st International Symposium on Mathematical Programming Berlin, vol Extra, Journal der Deutschen Mathematiker-Vereinigung, chap A Brief History of Linear and Mixed-Integer Programming Computation, pp 107–121

Burke E, Marecek J, Parkes A, Rudová H (2008) Uses and abuses of mip in course timetabling. In: Poster at the Workshop on Mixed Integer Programming, MIP2007, Montréal

Cacchiani V, Caprara A, Roberti R, Toth P (2013) A new lower bound for curriculum-based course timetabling. Computers & Operations Research 40(10):2466 – 2477

Caserta M, Voss S (2010) Metaheuristics: Intelligent problem solving. In: Maniezzo V, Statzle T, Voss S (eds) Matheuristics, Annals of Information Systems, vol 10, Springer US, pp 1–38

Danna E, Rothberg E, Le Pape C (2005) Exploring relaxation induced neighborhoods to improve mip solutions. Mathematical Programming 102(1):71–90

Di Gaspero L, McCollum B, Schaerf A (2007) The second international timetabling competition (itc-2007): Curriculum-based course timetabling (track 3). Tech. rep., School of Electronics, Electrical Engineering and Computer Science, Queenes University SARC Building, Belfast, United Kingdom

Fischetti M, Lodi A (2003) Local branching. Mathematical Programming 98:23–47

Hao JK, Benlic U (2011) Lower bounds for the ITC-2007 curriculum-based course timetabling problem. European Journal of Operational Research 212(3):464 – 472

Kiefer A, Hartl R, Schnell A (2014) Adaptive large neighborhood search for the curriculum-based course timetabling problem. Tech. rep., University of Vienna

Lach G, Lübbecke M (2012) Curriculum based course timetabling: new solutions to udine benchmark instances. Annals of Operations Research 194:255–272

Müller T (2009) Itc2007 solver description: a hybrid approach. Annals of Operations Research 172:429–446

Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. In: Maher M, Puget JF (eds) Principles and Practice of Constraint Programming CP98, Lecture Notes in Computer Science, vol 1520, Springer Berlin / Heidelberg, pp 417–431

Sniedovich M, Voß S (2006) The corridor method: a dynamic programming inspired metaheuristic. Control and Cybernetics 35(3):551

Vielma JP (2015) Mixed integer linear programming formulation techniques. SIAM Review 57(1):3–57